REMARKS

Reconsideration and allowance are respectfully requested.

Applicant notes with appreciation the Examiner's acknowledgement of the foreign priority claim. Submitted with this Amendment is a certified copy of the priority United Kingdom patent application as required by 35 U.S.C. §119(b). Acknowledgement of receipt of the certified copy is respectfully requested.

The Examiner objects to the disclosure contending that page 5, lines 25-29 repeat the previous paragraph. Applicant respectfully disagrees. The text at page 5, lines 25-29 does not repeat lines 20-24 in the previous paragraph on page 5, because lines 20-24 specify a threshold execution speed and lines 25-29 specify a threshold execution frequency. Withdrawal of the objection to the specification is requested.

The Examiner objects to claims 17 and 18 noting a misspelling. The misspelling has been corrected in both claims in this amendment. Withdrawal of the claim objections is respectfully requested.

Claims 8 and 9 stand rejected under 35 U.S.C. §112, second paragraph contending that the term "substantially" is unclear. It has been amended to remove the word "substantially" and now recites "each of a majority of interpreted code instruction words corresponds to a native code instruction word." This is a broadening rather than a narrowing amendment. No equivalents have been surrendered. Withdrawal of the rejection under 35 U.S.C. §112, second paragraph is respectfully requested.

Claims 1-5, 8 and 19 stand rejected under 35 U.S.C. §102(e) as allegedly being anticipated by U.S. 6,513,156 to Bak. This rejection is respectfully traversed.

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Bak fails to satisfy this rigorous standard.

Bak teaches a method for increasing an execution speed of virtual machine instructions for a "function." The term "function" is defined in column 4, lines 20-21 as a "software routine (also called a subroutine, procedure, member function, and method)." The problem addressed by Bak is that when an interpreter executes a bytecode program, it repeatedly executes execute, advance, and dispatch steps. Because each advance and dispatch step is executed for each virtual machine instruction, the interpreter's execution time may be quite slow. Bak attempts to improve the interpreter's performance by compiling a sequence of certain virtual machine instructions into native machine instructions called a "snippet" so that all of the advance and dispatch steps in the sequence may be removed except the last two. A new, interpreter instruction called "go_native" is used to "call" the snippet.

During execution of an interpreted program, the interpreter decides when to substitute a sequence of bytecodes with the snippet as shown in Figure 5. The management information of each snippet 307 includes storage for the original bytecode 2 which is overwritten by the go_native bytecode as well as the original address of bytecode 2. As a result, the original bytecode sequence may be restored when the snippet is removed.

In the Figure 5 example, snippet 307 includes native machine instructions for bytecodes 2-5 of the virtual machine instructions 301. As explained at column 8, lines 32-35:

> When the interpreter executes the go_native bytecode, the interpreter will look up the snippet in the snippet zone specified by the go_native bytecode and then activate the native machine instructions in the snippet.

Once the snippet native machine instructions have been executed, the "interpreter continues with the execution of bytecode 6 as if no snippet existed." Column 8, lines 38-39.

Bak's summarizes his invention in column 11, line 17 and following:

> The preceding has described how the invention utilizes dynamically generated native machine instructions for sequences of interpreted code so that a function may be more efficiently executed utilizing a hybrid of virtual and native machine instructions. The execution of an interpreted program can be significantly sped up because frequently used code sequences may be executed in native code rather than an interpreted fashion.

Thus, Bak is concerned with switching from execution of interpreted instructions to a subset of native code instructions. In contrast, the inventor of the present invention addressed the problem of enabling an efficient switch from execution of native code instructions to interpreted code instructions.

Regarding claim 1, the Examiner contends that "a native code portion invokes interpretation of an interpreted code portion by executing a native code call instruction to said instruction interpreter" is found in Figure 11 and column 13-17 in Bak. Applicant respectfully disagrees.

Figure 11 is a flowchart diagram outlining a jump from executing an interpreted instruction to a snippet of native code in order to implement that instruction—an invoke virtual bytecode. As explained at the top of column 12, line 2:

> At step 901, the system saves the current bytecode pointer so that the interpreter can continue at the right location after returning...The system pushes the interpreter return address on the stack at step 903. The interpreter return address is a predefined location where the execution of the interpreter from invoke_virtual bytecodes should resume.

Thus, Bak describes the opposite of what is recited in element (v) of claim 1. Bak's interpreted code is invoking a native code portion corresponding to the invoke snippet. Conversely, element (v) in claim 1 recites "a native code portion invokes interpretation of an interpreted code portion." Thus, element (v) of claim 1 is not disclosed in Bak.

Claim 1 also recites executing a native code call instruction to call interpreter software to interpret an interpreted instruction. In contrast, the go_native bytecode call is

an interpreter instruction call to execute the native instructions in the snippet. Thus, Bak lacks this additional claim feature.

Bak further fails to disclose in claim elements (vi) and (vii) triggering, in response to the native code call instruction, "generation of a return address specifying a location within said memory for said native code call instruction," where "said instruction interpreter uses said return address as a pointer to said interpreted code portion within said memory." In Figure 11, and column 12, lines 13-17, Bak discloses that an interpreter invoke-virtual bytecode, which is used to invoke a specified virtual function, is replaced by native machine instructions in which the address of the specified virtual function is hard-coded. Column 11, lines 47-65. Bak does not specify that the virtual function itself is replaced by native code. Replacement of the invoke-virtual bytecode with the native code portion allows the time consuming processing task of following multiple levels of functions to find the class that includes the functions specified by the invoke-virtual bytecode to be performed at a compilation stage. Thus, in Bak's system, generating the address of the virtual function to be executed is performed at the compilation stage (enabling hard coding of the address) rather than at execution of a native code call as recited in claim 1.

But the distinction is even more fundamental. Simply reading a saved bytecode pointer from memory "so the interpreter may continue where it left off," (Column 12, line 19), is different from generating a return address by executing a native code function call to call the interpreter software, (and not native code as in Bak), where the location of the

interpreted code instructions to be executed is provided to the interpreter via the return address of the native code call instruction. Bak teaches the native code using the stored bytecode pointer to allow the interpreter to continue after the native code snippet execution is done. Claim 1 uses the return address part of the native code call to facilitate the initial call to the interpreter. This feature is absent in Bak.

Lacking multiple features of independent claim 1, Applicant respectfully submits that the anticipation rejection based on Bak is improper and should be withdrawn. Claims 19 and 20 contain similar distinctions and likewise should also be allowed. Additional features associated with dependent claims 2-18 need not be addressed because the rejection of their independent claim 1 is improper.

The application is in condition for allowance. An early notice to that effect is earnestly solicited.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
John R. Lastova
Reg. No. 33,149

JRL:at
1100 North Glebe Road, 8th Floor
Arlington, VA 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100